283[Move Zeroes](https://leetcode.com/problems/move-zeroes/)49.9%Easy

class Solution(object):

def moveZeroes(self, nums):

"""

:type nums: List[int]

:rtype: void Do not return anything, modify nums in-place instead.

"""

insertIndex = 0

for i in nums:

if i:

nums[insertIndex] = i

insertIndex += 1

for j in xrange(insertIndex, len(nums)):

nums[j] = 0

# this solution takes up to O(n) in time complexity and O(1) in space, but a little bit more than n, since it

# traverses the array and move 0s in the end

# the above solution still needs n array writes, then we can reduce the write operation by only swapping when a

# non-zero element is found

#All elements before the slow pointer (lastNonZeroFoundAt) are non-zeroes.

#All elements between the current and slow pointer are zeroes.

class Solution2(object):

def moveZeroes(self, nums):

"""

:type nums: List[int]

:rtype: void Do not return anything, modify nums in-place instead.

"""

lastNonZeroIndex = 0

for i in xrange(len(nums)):

if nums[i] != 0:

nums[lastNonZeroIndex], nums[i] = nums[i], nums[lastNonZeroIndex]

lastNonZeroIndex += 1

325[Maximum Size Subarray Sum Equals k](https://leetcode.com/problems/maximum-size-subarray-sum-equals-k/) 42.3%Medium

class Solution(object):

def maxSubArrayLen(self, nums, k):

# fake entry to make sum consistent

prefix = {0 : -1} # key is cumulative and value is starting index

cumulative, longest = 0, 0

for i, val in enumerate(nums):

cumulative += val

if cumulative not in prefix:

prefix[cumulative] = i #only recording first occurance to make it max

if cumulative - k in prefix:

longest = max(longest, i - prefix[cumulative-k])

return longest

301[Remove Invalid Parentheses](https://leetcode.com/problems/remove-invalid-parentheses/)35.1%Hard

from collections import deque

class Solution(object):

def removeInvalidParentheses(self, s):

"""

:type s: str

:rtype: List[str]

"""

if self.isValid(s): # check empty or already valid string

return [s]

queue = deque([s])

visited = {s}

res = []

found = False

while queue:

comb = queue.popleft()

if self.isValid(comb):

res.append(comb)

found = True

if found:

continue # this will skip next part but pop out all str in queue

for i in xrange(len(comb)):

if comb[i] == "(" or comb[i] == ")": # only deals with parenthesis

newcomb = comb[:i] + comb[i+1:]

if newcomb not in visited:

queue.append(newcomb)

visited.add(newcomb) # don't for get this

return res

def isValid(self, s):

count = 0

for i in s:

if i == "(":

count += 1

elif i == ")":

count -= 1

if count < 0:

return False # ')' can not come first

return count == 0

67[Add Binary](https://leetcode.com/problems/add-binary/)32.5%Easy

class Solution(object):

def addBinary(self, a, b):

"""

:type a: str

:type b: str

:rtype: str

"""

if not a:

return b

if not b:

return a

if len(a) <= len(b): # let a be the longer one

a, b = b, a

carry = 0

res = ''

for i in xrange(1, len(b)+1): # add both digit on b

digit = (int(b[-i]) + int(a[-i]) + carry) % 2

carry = (int(b[-i]) + int(a[-i]) + carry) / 2

res = str(digit) + res

for j in xrange(len(b)+1, len(a)+1): # add exclusive digit on a

digit = (int(a[-j]) + carry) % 2

carry = (int(a[-j]) + carry) / 2

res = str(digit) + res

if carry == 1: # do not forget the left one

res = str(carry) + res

return res

311[Sparse Matrix Multiplication](https://leetcode.com/problems/sparse-matrix-multiplication/) 50.8%Medium

class Solution(object):

def multiply(self, A, B):

"""

:type A: List[List[int]]

:type B: List[List[int]]

:rtype: List[List[int]]

"""

if not A or not B:

return [[]]

m, n, l = len(A), len(A[0]), len(B[0])

if len(B) != n:

raise Exception("A's column number should be equal to B's column number")

res = [[0] \* l for i in xrange(m)]

for i in xrange(m):

for j in xrange(n):

if A[i][j] != 0:

for k in xrange(l):

if B[j][k] != 0:

res[i][k] += A[i][j] \* B[j][k]

return res

'''

this is a solution without using hashtable, so the basic idea is to traverse

every element in A and B matrix, but skip when either is 0, which saves a lot

in sparce matrix example

'''

# with table

class Solution(object):

def multiply(self, A, B):

"""

:type A: List[List[int]]

:type B: List[List[int]]

:rtype: List[List[int]]

"""

# use a table to store the non-zero num in B

tableB = {}

if len(A) == 0:

return []

m, n = len(A), len(A[0])

if len(B) != n:

raise Exception("A's col number is not equal to B's row number")

k = len(B[0])

res = [[0] \* k for i in xrange(m)] # res row num = A' row num, col num = B' col num

for i in xrange(n):

for j in xrange(k):

if B[i][j] != 0:

tableB[(i,j)] = B[i][j] # update table

# loop over A

for i in xrange(m):

for j in xrange(n):

if A[i][j] != 0:

for p in xrange(k):

if (j,p) in tableB:

res[i][p] += A[i][j] \* tableB[(j,p)]

return res

314[Binary Tree Vertical Order Traversal](https://leetcode.com/problems/binary-tree-vertical-order-traversal/) 36.5%Medium

from collections import defaultdict

class Solution(object):

def verticalOrder(self, root):

"""

:type root: TreeNode

:rtype: List[List[int]]

"""

if not root:

return []

dic = defaultdict(list) # not ordereddict, since it records insert order, not sorted keys

queue = []

queue.append((root, 0))

dic[0] = [root.val]

while queue:

levelnum = len(queue)

while levelnum:

node, index = queue.pop(0)

if node.left:

queue.append((node.left, index-1))

dic[index-1].append(node.left.val)

if node.right:

queue.append((node.right, index+1))

dic[index+1].append(node.right.val)

levelnum -= 1

res = []

for key in sorted(dic.keys()):

res.append(dic[key])

return res

'''

use bfs to traverse the tree and record the vertical index of the nodes (O(n))

use a dictionary to record the nodes value under different indexes, which takes O(nlog(n))

total: O(nlog(n))

'''

273[Integer to English Words](https://leetcode.com/problems/integer-to-english-words/)22.1%Hard

class Solution(object):

def numberToWords(self, num):

"""

:type num: int

:rtype: str

"""

# use split because it is easy to type than ['One', 'Two', ...]

lessThan20 = 'One Two Three Four Five Six Seven Eight Nine Ten ' \

'Eleven Twelve Thirteen Fourteen Fifteen Sixteen Seventeen Eighteen Nineteen'.split(" ")

tens = 'Ten Twenty Thirty Forty Fifty Sixty Seventy Eighty Ninety'.split(" ")

thousands = ' Thousand Million Billion'.split(" ") # space at the begining is important

def helper(num): # recursively handle the number < 1000

if not num:

return ""

elif num < 20:

return lessThan20[num - 1] + " "

elif num < 100:

return tens[num / 10 - 1] + " " + helper(num % 10)

return lessThan20[num / 100 - 1] + " Hundred " + helper(num % 100)

if not num:

return 'Zero'

words = ''

i = 0 # count the thousands

# calculate the num from the least important digits

while num > 0:

if num % 1000 > 0:

words = helper(num % 1000) + thousands[i] + " " + words

num /= 1000

i += 1

return words.strip()

17[Letter Combinations of a Phone Number](https://leetcode.com/problems/letter-combinations-of-a-phone-number/)34.6%Medium

from collections import deque

class Solution(object):

def letterCombinations(self, digits):

if not digits:

return []

mapping=[[''],['\_'],['a','b','c'],['d','e','f'],['g','h','i'],['j','k','l'],['m','n','o'],['p','q','r','s'],['t','u','v'],['w','x','y','z']]

queue = deque()

queue.append("")

for i in digits:

for j in xrange(len(queue)):

last = queue.popleft()

for k in mapping[int(i)]:

queue.append(last + k)

return list(queue)

'''

bfs solution avoids the entire copy of the last res

'''

278[First Bad Version](https://leetcode.com/problems/first-bad-version/)25.3%Easy

class Solution(object):

def firstBadVersion(self, n):

"""

:type n: int

:rtype: int

"""

i, j = 1, n

while i <= j:

if i == j: # edge case for equals

return i

mid = (i + j) / 2

if isBadVersion(mid):

if mid-1 > 0 and not isBadVersion(mid-1): # check backward

return mid

else:

j = mid

else: # not bad version

if mid+1 <= n and isBadVersion(mid+1): # check forward

return mid+1

else:

i = mid

'''

use binary search to check the bad one, whenever check the mid, check one more step forward or

backward to ensure if found the first bad version

'''

91[Decode Ways](https://leetcode.com/problems/decode-ways/)19.6%Medium

class Solution(object):

def numDecodings(self, s):

"""

:type s: str

:rtype: int

"""

if not s or s[0] == "0":

return 0

length = len(s)

res = [0] \* (length + 1) # memoization

res[0] = res[1] = 1 # initialize

for i in xrange(1, length):

if s[i] == "0":

if s[i-1] == "1" or s[i-1] == "2": # valid

res[i+1] = res[i-1]

else: # not valid

return 0

elif s[i-1] != "0" and int(s[i-1: i+1]) <= 26:

res[i+1] = res[i-1] + res[i]

else:# double digits not under 26, then only 1 option

res[i+1] = res[i]

return res[-1]

'''

The basic idea is dp, using an array to store all the previous combination numbers:

res[i+1] = res[i] + res[i-1] if (previous digits could be used) else res[i].

The trick part is dealing with 0, so if the digit before 0 is '1' or '2', count as

1 combination, the digit after 0 will only be considered as 1 combination

'''

10[Regular Expression Matching](https://leetcode.com/problems/regular-expression-matching/)24.1%Hard

class Solution(object):

def isMatch(self, s, p):

"""

:type s: str

:type p: str

:rtype: bool

"""

m, n = len(s), len(p)

dp = [[False] \* (n+1) for i in xrange(m+1)]

dp[0][0] = True

# initialize the first row

for i in xrange(n):

if p[i] == "\*" and dp[0][i-1]:

dp[0][i+1] = True

for i in xrange(m):

for j in xrange(n):

if p[j] == "." or p[j] == s[i]: #match for a certain char

dp[i+1][j+1] = dp[i][j]

if p[j] == "\*":

if p[j-1] != s[i] and p[j-1] != ".": #empty repeat

dp[i+1][j+1] = dp[i+1][j-1] # skip one char back to look up

else: #one or more repeat

dp[i+1][j+1] = dp[i+1][j] or dp[i][j+1] or dp[i+1][j-1]

return dp[m][n]

'''

1, If p.charAt(j) == s.charAt(i) : dp[i][j] = dp[i-1][j-1];

2, If p.charAt(j) == '.' : dp[i][j] = dp[i-1][j-1];

3, If p.charAt(j) == '\*':

here are two sub conditions:

1 if p.charAt(j-1) != s.charAt(i) : dp[i][j] = dp[i][j-2] //in this case, a\* only counts as empty

2 if p.charAt(i-1) == s.charAt(i) or p.charAt(i-1) == '.':

dp[i][j] = dp[i-1][j] //in this case, a\* counts as multiple a

or dp[i][j] = dp[i][j-1] // in this case, a\* counts as single a

or dp[i][j] = dp[i][j-2] // in this case, a\* counts as empty

'''

253[Meeting Rooms II](https://leetcode.com/problems/meeting-rooms-ii/) 38.8%Medium

import heapq

class interval(object):

def \_\_init\_\_(self, s=0, e=0):

self.start = s

self.end = e

class Solution(object):

def minmumMeetingRooms(self, intervals):

intervals = sorted(intervals, key = lambda x: x.start) # has to sort because we have to pick interval in chronological order

h = []

rooms = 0

for i in intervals:

if h:

if i.start < h[0]:

rooms += 1

else:

heapq.heappop(h) # pop the earlist time

heapq.heappush(h, i.end)

else: # h is empty

rooms += 1

heapq.heappush(h, i.end)

return rooms

15[3Sum](https://leetcode.com/problems/3sum/)21.6%Medium

class Solution(object):

def threeSum(self, nums):

"""

:type nums: List[int]

:rtype: List[List[int]]

"""

nums.sort()

res = []

for i in xrange(len(nums)-2):

if i == 0 or i != 0 and nums[i] != nums[i-1]: # remove duplicate first element

target = 0 - nums[i]

lo = i+1

hi = len(nums) - 1

while lo < hi:

if nums[lo] + nums[hi] == target:

res.append([nums[i], nums[lo], nums[hi]])

while lo < hi and nums[lo + 1] == nums[lo]: # avoid duplicate elements

lo += 1

while lo < hi and nums[hi - 1] == nums[hi]:

hi -= 1

lo += 1

hi -= 1

elif nums[lo] + nums[hi] < target:

lo += 1

else:

hi -= 1

return res

277[Find the Celebrity](https://leetcode.com/problems/find-the-celebrity/) 35.3%Medium

'''

2 pass solution, find the candidate in the first pass and validate it

in the second

'''

class Solution(object):

def findCelebrity(self, n):

"""

:type n: int

:rtype: int

"""

candidate = 0

for i in xrange(n):

if knows(candidate, i):

candidate = i

for i in xrange(n):

if i != candidate and knows(candidate, i) or not knows(i, candidate):

return -1

return candidate

158[Read N Characters Given Read4 II - Call multiple times](https://leetcode.com/problems/read-n-characters-given-read4-ii-call-multiple-times/) 24.1%Hard

# The read4 API is already defined for you.

# @param buf, a list of characters

# @return an integer

# def read4(buf):

class Solution(object):

def \_\_init\_\_(self):

self.queue = [] # used to save all the char fetched

def read(self, buf, n):

"""

:type buf: Destination buffer (List[str])

:type n: Maximum number of characters to read (int)

:rtype: The number of characters read (int)

"""

idx = 0

while True:

buf4 = [""] \* 4 # defines the size of return array, otherwise index out of range, contain the return chars, it is the destination, not the source

l = read4(buf4) # return the length, read4() function will go to the file and fetch chars

self.queue.extend(buf4)

leftnum = min(len(self.queue), n - idx) # number of chars left

for i in xrange(leftnum):

buf[idx] = self.queue.pop(0)

idx += 1

if leftnum == 0: # reach the end

break

return idx

200[Number of Islands](https://leetcode.com/problems/number-of-islands/)34.6%Medium

class Solution(object):

def numIslands(self, grid):

"""

:type grid: List[List[str]]

:rtype: int

"""

def dfs(i,j,grid):

if i < 0 or j < 0 or i >= rows or j >= cols or grid[i][j] != '1':

return

grid[i][j] = '0'

dfs(i-1, j, grid)

dfs(i, j+1, grid)

dfs(i, j-1, grid)

dfs(i+1, j, grid)

if not grid:

return 0

rows = len(grid)

cols = len(grid[0])

islands = 0

for i in xrange(rows):

for j in xrange(cols):

if grid[i][j] == "1":

dfs(i, j, grid)

islands += 1

return islands

297[Serialize and Deserialize Binary Tree](https://leetcode.com/problems/serialize-and-deserialize-binary-tree/)33.3%Hard

class Codec:

def serialize(self, root):

"""Encodes a tree to a single string.

:type root: TreeNode

:rtype: str

"""

queue = [root]

res = []

while queue:

node = queue.pop(0)

if node: #not null for node

queue.append(node.left)

queue.append(node.right)

res.append(str(node.val)) # turn to string

else:

res.append("#")

return " ".join(res) # not res.join("")

def deserialize(self, data):

"""Decodes your encoded data to tree.

:type data: str

:rtype: TreeNode

"""

if not data or data == "#":

return None

nodelist = data.split(" ")

root = TreeNode(nodelist.pop(0)) # set to the treenode

queue = [root]

while nodelist:

parent = queue.pop(0)

left = nodelist.pop(0)

left = None if left == "#" else TreeNode(int(left))

right = nodelist.pop(0)

right = None if right == "#" else TreeNode(int(right))

if left:

parent.left = left

queue.append(left)

if right:

parent.right = right

queue.append(right)

return root

'''

the basic idea is to use # to replace null in the string, when deserializing the string,

only add the not null value to the queue, which contains parents

'''

282[Expression Add Operators](https://leetcode.com/problems/expression-add-operators/)29.7%Hard

class Solution(object):

def addOperators(self, num, target):

"""

:type num: str

:type target: int

:rtype: List[str]

"""

if not num: # target could be 0

return []

res = []

self.helper(res, num, target, '', 0, 0)

return res

def helper(self,res, num, target, path, curval, multi):

if len(num) == 0 and curval == target:

res.append(path)

return

for i in xrange(1, len(num)+1): #i could reach len(num)

curStr = num[:i]

curNum = int(curStr)

if len(curStr) > 1 and curStr[0] == '0':

return

nextStr = num[i:]

if len(path) == 0: # do not insert operators in the front

self.helper(res, nextStr, target, curStr, curNum, curNum)

else:

self.helper(res, nextStr, target, path + '+' + curStr, curval + curNum, curNum)

self.helper(res, nextStr, target, path + '-' + curStr, curval - curNum, -curNum)

self.helper(res, nextStr, target, path + '\*' + curStr, curval - multi + curNum \* multi, multi\* curNum)

'''

1. use helper funtion to do the backtracking, and append new found to the res, not return any value

2. checking the leading 0 part, should not have multiple leading 0s

3. should save the number used for multiplication, since it changes the order

4. do not insert operator for the first char

'''

76[Minimum Window Substring](https://leetcode.com/problems/minimum-window-substring/)25.4%Hard

class Solution(object):

def minWindow(self, s, t):

"""

:type s: str

:type t: str

:rtype: str

"""

dic = {} # store the count of all the chars

targetcount = 0

sourcecount = 0

start = 0

minlen = float("inf")

minStr = ""

for i in t:

if i not in dic:

dic[i] = 1

else:

dic[i] += 1

targetcount += 1

# when loop through all the chars, subtract char's value if they are in the source

# but only add source count when they are needed. When the count matches, narrow the

# window size by moving start pointer

for i, val in enumerate(s):

if val in dic:

if dic[val] > 0: # means this char is needed

sourcecount += 1

dic[val] -= 1 # this shoud be subtracted anyway

while sourcecount == targetcount: # found all chars in s

if minlen >= i - start + 1:

minlen = min(minlen, i - start + 1)

minStr = s[start: i + 1]

if s[start] in dic:

dic[s[start]] += 1

if dic[s[start]] > 0:

sourcecount -= 1

start += 1

return minStr

257[Binary Tree Paths](https://leetcode.com/problems/binary-tree-paths/)38.5%Easy

class Solution(object):

def binaryTreePaths(self, root):

"""

:type root: TreeNode

:rtype: List[str]

"""

self.res = []

if root:

self.dfs(root, '')

return self.res

def dfs(self, root, path):

if not root.left and not root.right:

self.res.append(path + str(root.val))

if root.left:

self.dfs(root.left, path+str(root.val)+'->') # there are valid node left, put the arrow behind

if root.right:

self.dfs(root.right, path+str(root.val)+'->')

23[Merge k Sorted Lists](https://leetcode.com/problems/merge-k-sorted-lists/)27.2%Hard

import heapq

class Solution(object):

def mergeKLists(self, lists):

"""

:type lists: List[ListNode]

:rtype: ListNode

"""

if not lists:

return []

dummy = ListNode(0)

head = dummy

pq = []

for node in lists:

if node:

heapq.heappush(pq, (node.val, node))

while pq:

nextnode = heapq.heappop(pq)

n = nextnode[1]

head.next = n

head = head.next

if n.next:

n = n.next

heapq.heappush(pq, (n.val, n))

return dummy.next

161[One Edit Distance](https://leetcode.com/problems/one-edit-distance/) 31.2%Medium

class Solution(object):

def isOneEditDistance(self, s, t):

"""

:type s: str

:type t: str

:rtype: bool

"""

for i in xrange(min(len(s), len(t))):

if s[i] != t[i]:

if len(s) == len(t): # if they are equal, this char should be replaced

return s[i+1:] == t[i+1:]

elif len(s) > len(t): # if s is longer, the only possibility is deleting one char from s

return s[i+1:] == t[i:]

else:

return s[i:] == t[i+1:]

return abs(len(s) - len(t)) == 1 # check if there is only 1 difference left in the end

157[Read N Characters Given Read4](https://leetcode.com/problems/read-n-characters-given-read4/) 28.9%Easy

class Solution(object):

def read(self, buf, n):

"""

:type buf: Destination buffer (List[str])

:type n: Maximum number of characters to read (int)

:rtype: The number of characters read (int)

"""

idx = 0

while n > 0:

buf4 = [""] \* 4

l = read4(buf4)

if not l: # if no more char in file in

return idx

for i in xrange(min(l, n)):

buf[idx] = buf4[i]

idx += 1

n -= 1

return idx

211[Add and Search Word - Data structure design](https://leetcode.com/problems/add-and-search-word-data-structure-design/)22.9%Medium

class TreeNode(object):

def \_\_init\_\_(self, val):

self.val = val

self.isWord = False

self.children = {}

class WordDictionary(object):

def \_\_init\_\_(self):

self.root = TreeNode(0)

def addWord(self, word):

cur = self.root

for char in word:

if char not in cur.children:

cur.children[char] = TreeNode(char)

cur = cur.children[char] # skip the existed char

cur.isWord = True

def search(self, word):

def find(word, node):

if not word:

return node.isWord

char, word = word[0], word[1:] # first char and the rest

if char != '.':

return char in node.children and find(word, node.children[char])

return any(find(word, kid) for kid in node.children.values())

return find(word, self.root)

'''

A trie structure. In search, the idea is to use find function to skip a level if the char is '.'

'''

285[Inorder Successor in BST](https://leetcode.com/problems/inorder-successor-in-bst/) 36.1%Medium

# iterative

class Solution2(object):

def inorderSuccessor(self, root, p):

"""

:type root: TreeNode

:type p: TreeNode

:rtype: TreeNode

"""

if not root:

return None

res = None

while root:

if root.val <= p.val:

root = root.right

else:

res = root

root = root.left # this will continue the loop until the bigger val is found in root.left subtree and change res again

return res

139[Word Break](https://leetcode.com/problems/word-break/)29.9%Medium

class Solution(object):

def wordBreak(self, s, wordDict):

"""

:type s: str

:type wordDict: List[str]

:rtype: bool

"""

dp = [False] \* (len(s)+1)

dp[0] = True

for end in xrange(1, len(s)+1):

for start in xrange(0, end):

if dp[start] and s[start:end] in wordDict:

dp[end] = True

break

return dp[-1]

'''

dp[i] stands for whether subarray(0, i) can be segmented into words from the dictionary.

So dp[0] means whether subarray(0, 0) (which is an empty string) can be segmented, and of course the answer is yes.

The default value for boolean array is false. Therefore we need to set f[0] to be true.

'''

121[Best Time to Buy and Sell Stock](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/)41.3%Easy

class Solution(object):

def maxProfit(self, prices):

"""

:type prices: List[int]

:rtype: int

"""

if not prices:

return 0

smallest = prices[0] # record the previous smallest price

dp = [0] \* len(prices)

for i in xrange(1, len(prices)):

dp[i] = max(prices[i] - smallest, dp[i-1])

smallest = min(smallest, prices[i])

return dp[-1]

56[Merge Intervals](https://leetcode.com/problems/merge-intervals/)30.2%Medium

问题就是李寇伍溜拆开问：  
1) 合并2个intervals.1point3acres缃�  
2) 判断两个intervals 是否overlapping  
3) 李寇伍溜  
题目很简单，不过有些条件小哥并没告诉我，我问他他才说，嗯嗯这是我希望你问的。。。。例如，interval是开区间还是闭区间，[1,2][2,3]算不算overlap，input是什么样的数据结构，等。  
细节要求也挺多的，例如我写了lambda expression他问我如果不用lambda怎么写，deep copy的意义是啥，等等。

class Solution(object):

def merge(self, intervals):

"""

:type intervals: List[Interval]

:rtype: List[Interval]

"""

if len(intervals) == 0: return []

#sort the start point first

intervals = sorted(intervals, key = lambda x : x.start)

res = [intervals[0]]

#if there is overlap, merge 2 intervals; otherwise append to list

for i in xrange(1, len(intervals)):

if intervals[i].start <= res[-1].end:

res[-1].end = max(intervals[i].end, res[-1].end)

else:

res.append(intervals[i])

return res

341[Flatten Nested List Iterator](https://leetcode.com/problems/flatten-nested-list-iterator/)41.6%Medium

class NestedIterator(object):

def \_\_init\_\_(self, nestedList):

"""

Initialize your data structure here.

:type nestedList: List[NestedInteger]

"""

self.s = nestedList[:]

def next(self):

"""

:rtype: int

"""

return self.s.pop(0).getInteger()

def hasNext(self):

"""

:rtype: bool

"""

while self.s and not self.s[0].isInteger(): # while s is not empty and the first element is not integer, it must be a list

self.s = self.s.pop(0).getList() + self.s # pop the list out and add to the beginning of self.s

if self.s and self.s[0].isInteger():

return True

return False

1[Two Sum](https://leetcode.com/problems/two-sum/)34.7%Easy

class Solution(object):

def twoSum(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: List[int]

"""

if len(nums) <= 1:

return False

dic = {}

for i in range(len(nums)):

if nums[i] in dic:

return [i,dic[nums[i]]]

else:

compliment = target - nums[i]

dic[compliment] = i

173[Binary Search Tree Iterator](https://leetcode.com/problems/binary-search-tree-iterator/)41.4%Medium

class BSTIterator(object):

def \_\_init\_\_(self, root):

"""

:type root: TreeNode

"""

self.stack = []

self.pushAllLeft(root)

def hasNext(self):

"""

:rtype: bool

"""

return self.stack

def next(self):

"""

:rtype: int

"""

node = self.stack.pop()

self.pushAllLeft(node.right)

return node.val

def pushAllLeft(self, node): # push all the left node to the stack

while node:

self.stack.append(node)

node = node.left

'''

First push all the left nodes to the stack, and for each node popped, also push all the left node of its right one, it

is basically iterative in-order traversal of a tree

'''

125[Valid Palindrome](https://leetcode.com/problems/valid-palindrome/)26.3%Easy

'''

str.isalnum()

Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise.

'''

class Solution(object):

def isPalindrome(self, s):

"""

:type s: str

:rtype: bool

"""

s = s.lower()

if s == "":

return True

i = 0

j = len(s)-1

while i < j:

while i < j and not s[i].isalnum():

i += 1

while i < j and not s[j].isalnum():

j -= 1

if s[i] == s[j]:

i += 1

j -= 1

else:

return False

return True

# a pythonic way to solve

class Solution:

# @param s, a string

# @return a boolean

def isPalindrome(self, s):

s = filter(str.isalnum, s.lower())

return s == s[::-1]

75[Sort Colors](https://leetcode.com/problems/sort-colors/)38.0%Medium

class Solution(object):

def sortColors(self, nums):

"""

:type nums: List[int]

:rtype: void Do not return anything, modify nums in-place instead.

"""

n = len(nums)

if n <= 1:

return

left, i, right = 0, 0, n-1

while i <= right:

while nums[i] == 2 and i < right:

nums[i], nums[right] = nums[right], nums[i]

right -= 1

while nums[i] == 0 and i > left:

nums[i], nums[left] = nums[left], nums[i]

left += 1

i += 1

78[Subsets](https://leetcode.com/problems/subsets/)40.7%Medium

class Solution(object):

def subsets(self, nums):

"""

:type nums: List[int]

:rtype: List[List[int]]

"""

"""

every element can be or not be in a subset, so there are totally

2^n combinations. We can start from the 1st element and add lists that with

it or without it, and iterate each elements.

if the nums = [1,2,3] the process could be [[]] -> [[],[1]] -> [[],[1],[2],[1,2]]

-> [[],[1],[2],[1,2], [3],[1,3],[2,3],[1,2,3]]

"""

if not nums:

return [[]]

res = [[]]

index = 0

while index < len(nums):

res += [i + [nums[index]] for i in res]

index += 1

return res

98[Validate Binary Search Tree](https://leetcode.com/problems/validate-binary-search-tree/)23.3%Medium

iterative:

class Solution(object):

def isValidBST(self, root):

"""

:type root: TreeNode

:rtype: bool

"""

if not root:

return True

stack = []

pre = TreeNode(None)

while root or stack != []:

while root:

stack.append(root)

root = root.left

root = stack.pop()

if pre and root.val <= pre.val:

return False

pre = root

root = root.right

return True

recursive:

class Solution(object):

def isValidBST(self, root):

"""

:type root: TreeNode

:rtype: bool

"""

output = []

self.inOrder(root, output)

for i in range(1, len(output)):

if output[i] <= output[i-1]:

return False

return True

def inOrder(self, parent, output):

if not parent:

return

self.inOrder(parent.left, output)

output.append(parent.val)

self.inOrder(parent.right, output)

43[Multiply Strings](https://leetcode.com/problems/multiply-strings/)27.1%Medium

class Solution(object):

def multiply(self, num1, num2):

"""

:type num1: str

:type num2: str

:rtype: str

"""

if num1 == '0' or num2 == '0':

return '0'

m = len(num1)

n = len(num2)

res = [0] \* (m + n)

# Step 1: get all one-digit products

# Time complexity: O(n1\*n2)

for i in xrange(m-1, -1, -1):

for j in xrange(n-1, -1, -1):

mul = (ord(num1[i]) - ord("0")) \* (ord(num2[j]) - ord("0"))

p1 = i + j # the position index

p2 = i + j + 1

res[p1] += mul / 10

res[p2] += mul % 10

# Step 2: Sweep through the stored products with carry

# Time complexity: O(n1+n2)

carry = 0

for i in xrange(m+n-1, -1, -1):

temp = carry + res[i]

carry, res[i] = temp / 10, str(temp % 10)

return ''.join(res) if res[0] != '0' else ''.join(res[1:])

206[Reverse Linked List](https://leetcode.com/problems/reverse-linked-list/)45.5%Easy

# iterative

class Solution(object):

def reverseList(self, head):

"""

:type head: ListNode

:rtype: ListNode

"""

prev = None

curr = head

while curr:

nxt = curr.next

curr.next = prev

prev = curr

curr = nxt

return prev

# recursive

class Solution(object):

def reverseList(self, head):

"""

:type head: ListNode

:rtype: ListNode

"""

if not head or not head.next:

return head

p = self.reverseList(head.next)

head.next.next = head

head.next = None

return p

# p is the end node that is passed back level by level

# in the process, change the reference

133[Clone Graph](https://leetcode.com/problems/clone-graph/)25.1%Medium

class UndirectedGraphNode:

def \_\_init\_\_(self, x):

self.lable = x

self.neighbors = []

class Solution:

def cloneGraph(self, node):

if not node:

return None

queue = [] # used for bfs

map = {} # store node info, key is original node and value is the new node

queue.append(node)

# step 1: copy all the nodes

while queue:

curr = queue.pop(0)

copy = UndirectedGraphNode(curr.label)

map[curr] = copy

for n in curr.neighbors:

if n not in map: # skip all visited nodes

queue.append(n)

# step 2: copy connections between them

for ori in map.keys():

copy = map[ori]

for nei in ori.neighbors: # not copy.neighbors = ori.neighbours because we have to add new Nodes, rather than old ones

copy.neighbors.append(map[nei])

return map[node]

'''

step 1: copy all the nodes

step 2: copy connections between them

'''

252[Meeting Rooms](https://leetcode.com/problems/meeting-rooms/) 47.4%Easy

class Solution(object):

def canAttendMeetings(self, intervals):

"""

:type intervals: List[Interval]

:rtype: bool

"""

sort = sorted(intervals, key = lambda x : x.start)

for i in xrange(len(sort)-1):

if sort[i].end > sort[i+1].start:

return False

return True

218[The Skyline Problem](https://leetcode.com/problems/the-skyline-problem/)27.2%Hard

import heapq

class Solution(object):

def getSkyline(self, buildings):

"""

:type buildings: List[List[int]]

:rtype: List[List[int]]

"""

xs = []

# append all x coordinates

for l, r, h in buildings:

xs.append(l)

xs.append(r)

xs.sort()

result, activeheap, i = [], [], 0

for x in xs:

#if the rect.end < x, the rect has been checked, romove the rect

while activeheap and activeheap[0][1] <= x:

heapq.heappop(activeheap)

#add starting building

while i < len(buildings) and buildings[i][0]==x:

#in order to maintain max-heap, height is changed to negative

heapq.heappush(activeheap, (-buildings[i][2], buildings[i][1]))

i += 1

#y is either 0 or max height

y = 0 if not activeheap else -activeheap[0][0]

#if y changes

if not result or result[-1][1] != y: # if there is no previous building or change

result.append([x, y])

return result

'''

so the basic idea is:

step 1: push all the start and end point to the array and sort

step 2: loop through the array:

1.if meet the start point, push (height, end point) to the heap, # putting height in the front to sort

2.if meet the end point, check if it is already behind, otherwise

3.check the max height everytime, add to the result only if y changes

'''

236[Lowest Common Ancestor of a Binary Tree](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/)29.8%Medium

class Solution(object):

def lowestCommonAncestor(self, root, p, q):

"""

:type root: TreeNode

:type p: TreeNode

:type q: TreeNode

:rtype: TreeNode

"""

if root == p or root == q or not root:

return root

left = self.lowestCommonAncestor(root.left, p, q)

right = self.lowestCommonAncestor(root.right, p, q)

if left != None and right != None:

return root

else:

return left if left else right

'''

If the current (sub)tree contains both p and q, then the function result is their LCA.

If only one of them is in that subtree, then the result is that one of them.

If neither are in that subtree, the result is null/None/nil.

'''

49[Group Anagrams](https://leetcode.com/problems/group-anagrams/)34.8%Medium

class Solution(object):

def groupAnagrams(self, strs):

"""

:type strs: List[str]

:rtype: List[List[str]]

"""

if not strs:

return [[]]

dic = dict()

for i in strs:

comb = "".join(sorted(i))

if dic.get(comb):

dic[comb].append(i)

else:

dic[comb] = [i]

return dic.values()

time: O(N\* K log(k))

**class** **Solution**:

**def** **groupAnagrams**(strs):

ans **=** collections**.**defaultdict(list)

**for** s **in** strs:

count **=** [0] **\*** 26

**for** c **in** s:

count[ord(c) **-** ord('a')] **+=** 1

ans[tuple(count)]**.**append(s)

**return** ans**.**values()

time: O(N \* k)

79[Word Search](https://leetcode.com/problems/word-search/)26.8%Medium

class Solution(object):

def exist(self, board, word):

"""

:type board: List[List[str]]

:type word: str

:rtype: bool

"""

# loop through the grid, find the char that matches the first char of word, then

# dfs its neighbors to find the next char

if not board:

return False

m = len(board)

n = len(board[0])

visited = [[False] \* n for i in xrange(m)]

for i in xrange(m):

for j in xrange(n):

if self.dfs(i, j, board, word, visited):

return True

return False

def dfs(self, i, j, board, word, visited):

if len(word) == 0: # found all chars on board

return True

if (i < 0 or i >= len(board) or j < 0 or j >= len(board[0]) or

visited[i][j] or board[i][j] != word[0]): # out of bound or visited or not equal

return False

visited[i][j] = True # match the char

res = (self.dfs(i+1, j, board, word[1:], visited) # check 4 directions

or self.dfs(i, j+1, board, word[1:], visited)

or self.dfs(i-1, j, board, word[1:], visited)

or self.dfs(i, j-1, board, word[1:], visited))

visited[i][j] = False # recover for next round dfs

return res

238[Product of Array Except Self](https://leetcode.com/problems/product-of-array-except-self/)49.1%Medium

class Solution(object):

def productExceptSelf(self, nums):

"""

:type nums: List[int]

:rtype: List[int]

"""

# since we can not use division, we need to record the privous product

# we can sweep the array from the beginning and the end, and get

# the product below this index and above this index

output = []

p = 1

for i in xrange(len(nums)):

output.append(p) # save the product below the index

p \*= nums[i] # get product

p = 1

for i in xrange(len(nums)-1, -1, -1):

output[i] \*= p # multiply the product above the index

p \*= nums[i]

return output

38[Count and Say](https://leetcode.com/problems/count-and-say/)34.7%Easy

class Solution(object):

def countAndSay(self, n):

"""

:type n: int

:rtype: str

"""

res = '1'

for i in xrange(n-1):

temp = ''

count = 1

for j, val in enumerate(res):

if j < len(res)-1 and val == res[j+1]: # duplicate

count += 1

else: # add and recover

temp += str(count) + val

count = 1

res = temp

return res

57[Insert Interval](https://leetcode.com/problems/insert-interval/)27.7%Hard

# Definition for an interval.

# class Interval(object):

# def \_\_init\_\_(self, s=0, e=0):

# self.start = s

# self.end = e

class Solution(object):

def insert(self, intervals, newInterval):

"""

:type intervals: List[Interval]

:type newInterval: Interval

:rtype: List[Interval]

"""

#init

if not intervals:

return [newInterval]

if not newInterval:

return intervals

res = []

pos = 0

#traverse the list, merge if overlapping, insert the newInterval in the end

for inter in intervals:

if inter.end < newInterval.start:

res.append(inter)

pos += 1

elif inter.start > newInterval.end:

res.append(inter)

else:

newInterval.start = min(inter.start, newInterval.start)

newInterval.end = max(inter.end, newInterval.end)

res.insert(pos, newInterval)

return res

#time: O(n)

#space: O(1)

209[Minimum Size Subarray Sum](https://leetcode.com/problems/minimum-size-subarray-sum/)30.7%Medium

class Solution(object):

def minSubArrayLen(self, s, nums):

"""

:type s: int

:type nums: List[int]

:rtype: int

"""

res = float('inf')

start = 0

Sum = 0

for i in xrange(len(nums)):

Sum += nums[i]

while Sum >= s and start <= i:

res = min(res, i-start+1)

Sum -= nums[start]

start += 1

if res == float('inf'):

return 0

else:

return res

215[Kth Largest Element in an Array](https://leetcode.com/problems/kth-largest-element-in-an-array/)39.2%Medium

class Solution(object):

def findKthLargest(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: int

"""

if not nums:

return -1

n = len(nums)

return self.quickSelect(nums, 0, n - 1, k)

# select kth largest element

def quickSelect(self, nums, start, end, k):

if start == end:

return nums[start]

left = start

right = end

pivot = nums[end] # choose the last num as pivot

while left <= right:

while left <= right and nums[left] > pivot: # put the bigger num on the left

left += 1

while left <= right and nums[right] < pivot:

right -= 1

# swap left and right

nums[left], nums[right] = nums[right], nums[left]

left += 1

right -= 1

# left == right here

if start + k - 1 <= right: # on the larger part

return self.quickSelect(nums, start, right, k)

elif start + k - 1 >= left: # on the smaller part

return self.quickSelect(nums, left, end, k - (left - start))

# found k th number

return nums[right + 1]

71[Simplify Path](https://leetcode.com/problems/simplify-path/)25.3%Medium

class Solution(object):

def simplifyPath(self, path):

"""

:type path: str

:rtype: str

"""

stack = []

for i in path.split('/'): # this could remove '//' case

if i == '..': # go back to upper level

if stack:

stack.pop()

elif i != '' and i != '.': # not empty and not equals to current dir

stack.append(i)

return '/' + '/'.join(stack) # join function will put '/' in between

# /... is a valid path

146[LRU Cache](https://leetcode.com/problems/lru-cache/)17.8%Hard

class ListNode(object):

def \_\_init\_\_(self, k, v):

self.key = k

self.value = v

self.prev = None

self.next = None

class LRUCache(object):

def \_\_init\_\_(self, capacity):

"""

:type capacity: int

"""

self.capacity = capacity

self.dic = dict()

self.head = ListNode(0, 0)

self.tail = ListNode(0, 0)

self.head.next = self.tail

self.tail.prev = self.head

def get(self, key):

"""

:type key: int

:rtype: int

"""

if key in self.dic:

n = self.dic[key]

self.\_remove(n)

self.\_add(n)

return n.value

return -1

def put(self, key, value):

"""

:type key: int

:type value: int

:rtype: void

"""

if key in self.dic:

self.\_remove(self.dic[key]) # remove it if existed

n = ListNode(key, value)

self.\_add(n) # add to the tail

self.dic[key] = n

if len(self.dic) > self.capacity: # remove head

n = self.head.next

self.\_remove(n)

del self.dic[n.key]

def \_remove(self, node):

p = node.prev

n = node.next

p.next = n

n.prev = p

def \_add(self, node):

p = self.tail.prev

p.next = node

self.tail.prev = node

node.prev = p

node.next = self.tail

13[Roman to Integer](https://leetcode.com/problems/roman-to-integer/)45.8%Easy

class Solution(object):

def romanToInt(self, s):

d = {'I':1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500, 'M':1000}

#the last digit will be added anyway

r = d[s[len(s)-1]]

print r

for i in range(len(s)-1):

if d[s[i]] < d[s[i+1]]:

r -= d[s[i]]

else:

r += d[s[i]]

return r

class Solution(object):

def search(self, arr, target):

"""

:type nums: List[int]

:type target: int

:rtype: int

"""

if len(arr) == 0:

return -1

lo, hi = 0, len(arr)-1

while lo < hi:

mid = (lo + hi) / 2

if arr[mid] == target:

return mid

if arr[lo] <= arr[mid]: # if the left is sorted

if arr[lo] <= target and target < arr[mid]: # in the left, target could equal to arr[lo]

hi = mid - 1

else: # in the right

lo = mid + 1

else: # the right is sorted

if arr[mid] < target and target <= arr[hi]:

lo = mid + 1

else:

hi = mid - 1

return lo if arr[lo] == target else -1 # lo == right

'''

compare the mid value, if not found, check if the left subarray is sorted

if sorted, check whether the target is in this interval by comparing the target with

lo and hi part, do the same thing for right part

'''

208[Implement Trie (Prefix Tree)](https://leetcode.com/problems/implement-trie-prefix-tree/)28.3%Medium

class TreeNode(object):

def \_\_init\_\_(self, val):

self.children = dict() #key is char, value is node

self.isWord = False

self.val = val

class Trie(object):

def \_\_init\_\_(self):

"""

Initialize your data structure here.

"""

self.root = TreeNode(0)

def insert(self, word):

"""

Inserts a word into the trie.

:type word: str

:rtype: void

"""

cur = self.root

for letter in word:

if not cur.children.get(letter):

cur.children[letter] = TreeNode(letter)

cur = cur.children[letter]

cur.isWord = True

def search(self, word):

"""

Returns if the word is in the trie.

:type word: str

:rtype: bool

"""

cur = self.root

for letter in word:

cur = cur.children.get(letter)

if not cur:

return False

return cur.isWord

def startsWith(self, prefix):

"""

Returns if there is any word in the trie that starts with the given prefix.

:type prefix: str

:rtype: bool

"""

cur = self.root

for letter in prefix:

cur = cur.children.get(letter)

if not cur:

return False

return True

90[Subsets II](https://leetcode.com/problems/subsets-ii/)36.3%Medium

class Solution2(object):

def subsetsWithDup(self, nums):

"""

:type nums: List[int]

:rtype: List[List[int]]

"""

nums.sort()

res = [[]]

for i in nums:

for j in res[:]:

if ([i] + j) not in res:

res += [[i] + j]

return res

44[Wildcard Matching](https://leetcode.com/problems/wildcard-matching/)20.1%Hard

class Solution(object):

def isMatch(self, s, p):

"""

:type s: str

:type p: str

:rtype: bool

"""

s\_cur, p\_cur, match, star = 0, 0, 0, -1

while s\_cur < len(s):

# match or "?"

if p\_cur < len(p) and (s[s\_cur] == p[p\_cur] or p[p\_cur] == "?"):

s\_cur += 1

p\_cur += 1

# if "\*"

elif p\_cur < len(p) and p[p\_cur] == "\*":

star = p\_cur

match = s\_cur

p\_cur += 1

# if we do not have a match but have a "\*" before it, we record the

# next char that is not '\*', and find next match char in s

elif star != -1:

match += 1

s\_cur = match

p\_cur = star+1

else:

return False

#if there are "\*\*\*" at the end of pattern

while p\_cur < len(p) and p[p\_cur] == "\*":

p\_cur += 1

#reach the end

if p\_cur == len(p):

return True

else:

return False

'''

nots:

1. remember to check p\_cur < len(p) after adding p\_cur

'''

'''Analysis:

For each element in s

If \*s==\*p or \*p == ? which means this is a match, then goes to next element s++ p++.

If p=='\*', this is also a match, but one or many chars may be available, so let us save this \*'s position and the matched s position.

If not match, then we check if there is a \* previously showed up,

if there is no \*, return false;

if there is an \*, we set current p to the next element of \*, and set current s to the next saved s position.

e.g.

abed

?b\*d\*\*

a=?, go on, b=b, go on,

e=\*, save \* position star=3, save s position ss = 3, p++

e!=d, check if there was a \*, yes, ss++, s=ss; p=star+1

d=d, go on, meet the end.

check the rest element in p, if all are \*, true, else false;'''

128[Longest Consecutive Sequence](https://leetcode.com/problems/longest-consecutive-sequence/)36.8%Hard

class Solution(object):

def longestConsecutive(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

res = 0

dic = {}

for i in nums:

if i not in dic: # only check for unvisited nums

left = dic.get(i-1, 0)

right = dic.get(i+1, 0)

length = left + right + 1

res = max(length, res)

dic[i] = length

# only need update border because we only check unvisited nums

dic[i-left] = length # left border

dic[i+right] = length # right border

return res

'''

use a dictionary to track length up to current digit, and update left and right border

every time

'''

102[Binary Tree Level Order Traversal](https://leetcode.com/problems/binary-tree-level-order-traversal/)39.8%Medium

# Definition for a binary tree node.

# class TreeNode(object):

# def \_\_init\_\_(self, x):

# self.val = x

# self.left = None

# self.right = None

from collections import deque

class Solution(object):

def levelOrder(self, root):

"""

:type root: TreeNode

:rtype: List[List[int]]

"""

if root == None:

return []

queue = deque()

queue.append(root)

res = []

while queue:

size = len(queue) # level size

temp = []

while size:

node = queue.popleft()

temp.append(node.val)

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

size -= 1

res.append(temp)

return res

88[Merge Sorted Array](https://leetcode.com/problems/merge-sorted-array/)32.0%Easy

class Solution(object):

def merge(self, nums1, m, nums2, n):

"""

:type nums1: List[int]

:type m: int

:type nums2: List[int]

:type n: int

:rtype: void Do not return anything, modify nums1 in-place instead.

"""

k = m + n - 1

# start from the end and put the larger number first

while m > 0 and n > 0:

if nums1[m-1] > nums2[n-1]:

nums1[k] = nums1[m-1]

m -= 1

else:

nums1[k] = nums2[n-1]

n -= 1

k -= 1

if n > 0: # if nums in nums1 have been used and nums2 has numbers left

nums1[:n] = nums2[:n]

380[Insert Delete GetRandom O(1)](https://leetcode.com/problems/insert-delete-getrandom-o1/)39.1%Medium

import random

class RandomizedSet(object):

#use list to store the value, and use hashtable to keep track of the position

def \_\_init\_\_(self):

"""

Initialize your data structure here.

"""

self.vals = []

self.pos = {}

def insert(self, val):

"""

Inserts a value to the set. Returns true if the set did not already contain the specified element.

:type val: int

:rtype: bool

"""

if val not in self.pos:

self.vals.append(val)

self.pos[val] = len(self.vals)-1

return True

else:

return False

def remove(self, val):

"""

Removes a value from the set. Returns true if the set contained the specified element.

:type val: int

:rtype: bool

"""

if val in self.pos:

#exchange the positon of val and last element, then pop

last, index = self.vals[-1], self.pos[val]

self.pos[last], self.vals[index] = index, last

self.vals.pop()

self.pos.pop(val)

return True

else:

return False

def getRandom(self):

"""

Get a random element from the set.

:rtype: int

"""

return self.vals[random.randint(0, len(self.vals)-1)]

'''

the tricky part is to make remove function O(1), in order to do that,

swap the element and the last in self.val, and also change index, then pop the last one

and the index in the dictionary

'''

28[Implement strStr()](https://leetcode.com/problems/implement-strstr/)28.1%Easy

basically implement index() api.

class Solution(object):

def strStr(self, haystack, needle):

"""

:type haystack: str

:type needle: str

:rtype: int

"""

if not needle:

return 0

for i in xrange(len(haystack)-len(needle)+1): # has to plus 1 because we have to contain the first char of needle

for j in xrange(len(needle)):

if haystack[i+j] != needle[j]:

break

if j == len(needle)-1:

return i

return -1

377[Combination Sum IV](https://leetcode.com/problems/combination-sum-iv/)41.9%Medium

class Solution(object):

def combinationSum4(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: int

"""

d = {0 : 1} # map to store used target

return self.helper(nums, target, d)

def helper(self, nums, target, d):

if target in d: # if existed, no need to do it agian

return d[target]

res = 0

for i in nums:

if target >= i:

res += self.helper(nums, target - i, d)

d[target] = res

return res

s = Solution()

print s.combinationSum4([1,2,3], 4)

'''

if target has only one number left to reach, so loop through the nums to check the state:

in this example, comb(4) = comb(4-1) + comb(4-2) +comb(4-3)

since comb(0) = 1, we can store that in a map and start dp.

follow up:

The problem with negative numbers is that now the combinations could be potentially of infinite length.

Think about nums = [-1, 1] and target = 1. We can have all sequences of arbitrary length that follow the patterns

-1, 1, -1, 1, ..., -1, 1, 1 and 1, -1, 1, -1, ..., 1, -1, 1 (there are also others, of course, just to give an example).

So we should limit the length of the combination sequence, so as to give a bound to the problem.

'''

334[Increasing Triplet Subsequence](https://leetcode.com/problems/increasing-triplet-subsequence/)39.2%Medium

class Solution(object):

def increasingTriplet(self, nums):

"""

:type nums: List[int]

:rtype: bool

"""

c1, c2 = float('inf'), float('inf')

for i in nums:

if i <= c1:

c1 = i # smallest so far (it's a candidate for 1st element)

elif i <= c2: # c1< i < c2

c2 = i

else: # here when we have/had c1 < c2 already and i > c2

return True # the increasing subsequence of 3 elements exists

return False

127[Word Ladder](https://leetcode.com/problems/word-ladder/)19.4%Medium

from collections import deque

class Solution(object):

def ladderLength(self, beginWord, endWord, wordList):

"""

:type beginWord: str

:type endWord: str

:type wordList: List[str]

:rtype: int

"""

existed = set()

queue = deque()

queue.append(beginWord)

existed.add(beginWord)

length = 1

while queue:

size = len(queue)

while size:

word = queue.popleft()

if word == endWord:

return length

for i in xrange(len(word)):

for char in 'abcdefghijklmnopqrstuvwxyz':

string = word[:i] + char + word[i+1:]

if string in wordList and string not in existed:

queue.append(string)

existed.add(string)

size -= 1

length += 1

#no match

return 0

398[Random Pick Index](https://leetcode.com/problems/random-pick-index/)42.8%Medium

import random

class Solution(object):

def \_\_init\_\_(self, nums):

"""

:type nums: List[int]

:type numsSize: int

"""

self.nums = nums

def pick(self, target):

"""

:type target: int

:rtype: int

"""

result = -1

count = 0

for i in xrange(len(self.nums)):

if self.nums[i] == target:

if random.randint(0, count) == 0: # chance of 1/count

result = i #it will gaurantee that the first occurrence will be selected since randint(0, 0) = 0

count += 1

return result

50[Pow(x, n)](https://leetcode.com/problems/powx-n/)26.3%Medium

class Solution(object):

def myPow(self, x, n):

"""

:type x: float

:type n: int

:rtype: float

"""

if n == 0:

return 1

if n == 1:

return x

if n < 0:

n = -n

x = 1 / x

if n % 2 == 0:

return self.myPow(x \* x, n / 2)

if n % 2 == 1:

return self.myPow(x \* x, n / 2) \* x

269[Alien Dictionary](https://leetcode.com/problems/alien-dictionary/) 24.1%Hard

494[Target Sum](https://leetcode.com/problems/target-sum/)43.5%Medium

class Solution(object):

def findTargetSumWays(self, nums, S):

"""

:type nums: List[int]

:type S: int

:rtype: int

"""

if not nums:

return 0

if nums[0] == 0:

dic = {0:2} # +0 and -0

else:

dic = {nums[0]:1, -nums[0]:1}

for i in xrange(1, len(nums)):

nextdic = {}

for key in dic:

nextdic[key + nums[i]] = nextdic.get(key + nums[i], 0) + dic[key] # calculated + # of steps to get here

nextdic[key - nums[i]] = nextdic.get(key - nums[i], 0) + dic[key]

dic = nextdic # update

return dic.get(S, 0)

'''

update dic every time and save all the overlapped result in the same key for next use

time : O(n^2)

space : O(n^2)

'''

20[Valid Parentheses](https://leetcode.com/problems/valid-parentheses/)33.4%Easy

class Solution(object):

def isValid(self, s):

"""

:type s: str

:rtype: bool

"""

stack = []

for i in s:

if i == "(" or i == "{" or i == "[":

stack.append(i)

elif i == ")":

if not stack or stack[-1] != "(":

return False

stack.pop()

elif i == "}":

if not stack or stack[-1] != "{":

return False

stack.pop()

elif i == "]":

if not stack or stack[-1] != "[":

return False

stack.pop()

return len(stack) == 0

235[Lowest Common Ancestor of a Binary Search Tree](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/)39.0%Easy

class Solution(object):

def lowestCommonAncestor(self, root, p, q):

"""

:type root: TreeNode

:type p: TreeNode

:type q: TreeNode

:rtype: TreeNode

"""

if p.val < root.val and q.val < root.val:

return self.lowestCommonAncestor(root.left, p, q)

elif p.val > root.val and q.val > root.val:

return self.lowestCommonAncestor(root.right, p, q)

else: # either n1 and n2 are on different side or one of them is root

return root

286[Walls and Gates](https://leetcode.com/problems/walls-and-gates/) 44.1%Medium

from collections import deque

class Solution(object):

def wallsAndGates(self, rooms):

"""

:type rooms: List[List[int]]

:rtype: void Do not return anything, modify rooms in-place instead.

"""

inf = 2 \*\* 31 - 1

r = [0, 0, 1, -1]

c = [1, -1, 0, 0]

if not rooms:

return

queue = deque()

m = len(rooms)

n = len(rooms[0])

for row in xrange(m):

for col in xrange(n):

if rooms[row][col] == 0:

queue.append((row, col)) # record all gates position

while queue:

row, col = queue.popleft()

for i in xrange(4): # check 4 directions

n\_r = row + r[i] # new row

n\_c = col + c[i] # new col

if n\_r >= 0 and n\_r < m and n\_c >= 0 and n\_c < n:# inside matrix

if rooms[n\_r][n\_c] == inf:

rooms[n\_r][n\_c] = rooms[row][col] + 1

queue.append((n\_r, n\_c))

'''

Start from gate, push visited position to the queue and do the bfs. We do not need to update the

matrix when we reach here from the second gate is because the first reached range will always be

the smallest

'''

69[Sqrt(x)](https://leetcode.com/problems/sqrtx/)27.8%Easy

class Solution(object):

def mySqrt(self, x):

"""

:type x: int

:rtype: int

"""

def bs(lo, hi):

if lo+1 == hi:

return lo

mid = (lo+hi)/2

if mid \* mid < x:

return bs(mid, hi)

if mid \* mid > x:

return bs(lo, mid)

else:

return mid

return bs(1, x)

117[Populating Next Right Pointers in Each Node II](https://leetcode.com/problems/populating-next-right-pointers-in-each-node-ii/)33.8%Medium

class Solution:

# @param root, a tree link node

# @return nothing

def connect(self, root):

while root:

head = TreeLinkNode(0)

cur = head

#traverse a level

while root:

if root.left:

cur.next = root.left # assign head.next to next level's first node

cur = cur.next

if root.right:

cur.next = root.right

cur = cur.next

root = root.next # move pointer from the same level

#head.next will always point to the leftmost node in next level

#because cur moves itself forward but change the head.next pointer first

root = head.next

68[Text Justification](https://leetcode.com/problems/text-justification/)19.1%Hard

class Solution(object):

def fullJustify(self, words, maxWidth):

"""

:type words: List[str]

:type maxWidth: int

:rtype: List[str]

"""

# cur contains all the words in one row, but can also be used for counting needed spaces

res, cur, num\_of\_letters = [], [], 0

for w in words:

if num\_of\_letters + len(w) + len(cur) > maxWidth: # get enough words in one row

for i in xrange(maxWidth - num\_of\_letters):

cur[i % (len(cur)-1 or 1)] += ' ' # round robin, use mod to choose the position to add space

res.append(''.join(cur)) # finish inserting spaces

cur, num\_of\_letters = [], 0

cur += [w]

num\_of\_letters += len(w)

return res + [' '.join(cur).ljust(maxWidth)] # the last cur

# https://discuss.leetcode.com/topic/25970/concise-python-solution-10-lines

221[Maximal Square](https://leetcode.com/problems/maximal-square/)28.9%Medium

class Solution(object):

def maximalSquare(self, matrix):

"""

:type matrix: List[List[str]]

:rtype: int

"""

if not matrix:

return 0

m = len(matrix)

n = len(matrix[0])

dp = [[0] \* (n+1) for i in xrange(m+1)] # initialize with 1 more length to contain every elements

maxlen = 0

for i in xrange(1, m+1): # m+1 not m

for j in xrange(1, n+1):

if matrix[i-1][j-1] == '1':

dp[i][j] = min(dp[i-1][j-1], dp[i-1][j], dp[i][j-1]) + 1

maxlen = max(maxlen, dp[i][j])

return maxlen \* maxlen

'''

check if previous diagonal string is 1, then save min(previous neigbours) + 1

'''

274[H-Index](https://leetcode.com/problems/h-index/)33.2%Medium

class Solution(object):

def hIndex(self, citations):

"""

:type citations: List[int]

:rtype: int

"""

n = len(citations)

citations.sort()

hindex = 0

for i in xrange(n):

hindex = max(hindex, min(citations[i], n - i))

return hindex

'''

After sorting in descending order, h-index is the length of the largest square in the histogram

'''

85[Maximal Rectangle](https://leetcode.com/problems/maximal-rectangle/)28.0%Hard

210[Course Schedule II](https://leetcode.com/problems/course-schedule-ii/)28.0%Medium

from collections import defaultdict

class Solution(object):

def findOrder(self, numCourses, prerequisites):

"""

:type numCourses: int

:type prerequisites: List[List[int]]

:rtype: List[int]

"""

def dfs(graph, visited, i, res):

if visited[i] == -1: # has cycle

return False

if visited[i] == 1: # has visited

return True

visited[i] = -1 # is being visited

for j in graph[i]:

if not dfs(graph, visited, j, res):

return False

# reach the end

res.append(i)

visited[i] = 1

return True

graph = defaultdict(list)

res = []

# build the graph

for i in prerequisites:

graph[i[0]].append(i[1]) # value is key's prerequisite

visited = [0] \* numCourses

for i in xrange(numCourses):

if not dfs(graph, visited, i, res): # dfs here will append candidate node to visited

return []

return res

'''

implement topological sort using dfs, use flag to check visited or cyclic

'''

25[Reverse Nodes in k-Group](https://leetcode.com/problems/reverse-nodes-in-k-group/)30.8%Hard

234[Palindrome Linked List](https://leetcode.com/problems/palindrome-linked-list/)32.7%Easy

class Solution():

def isPalindrome(self, head):

"""

:type head: ListNode

:rtype: bool

"""

if head is None or head.next is None:

return True

fast = head

slow = head

# 1->2->3->2->1->None, fast = 5, slow = 3

#1->2->3->4->2->1->None, fast = None, slow = 4

# if the # of nodes is odd, slow is the middle one, if the

while fast and fast.next:

fast = fast.next.next

slow = slow.next

slow = self.reverseList(slow)

while slow:

if slow.val != head.val:

return False

head = head.next

slow = slow.next

return True

def reverseList(self, head):

newHead = None

while head:

Next = head.next

head.next = newHead

newHead = head

head = Next

return newHead

'''

reverse from the middle and start checking

'''

653[Two Sum IV - Input is a BST](https://leetcode.com/problems/two-sum-iv-input-is-a-bst/)50.9%Easy

from collections import deque

class Solution(object):

def findTarget(self, root, k):

"""

:type root: TreeNode

:type k: int

:rtype: bool

"""

s = set()

queue = deque()

queue.append(root)

while queue:

node = queue.popleft()

if k - node.val in s:

return True

s.add(node.val)

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

return False

642[Design Search Autocomplete System](https://leetcode.com/problems/design-search-autocomplete-system/) 29.2%Hard

639[Decode Ways II](https://leetcode.com/problems/decode-ways-ii/)22.8%Hard

class Solution(object):

def numDecodings(self, s):

"""

:type s: str

:rtype: int

"""

m = 10\*\*9 + 7

if not s or s[0] == "0":

return 0

length = len(s)

dp = [0] \* (length + 1)

dp[0] = 1

dp[1] = 9 if s[0] == "\*" else 1

for i in xrange(1, length):

if s[i] == "\*": # 1\*

dp[i+1] = 9 \* dp[i] # count as single digit

if s[i-1] == '1':

dp[i+1] = (dp[i+1] + dp[i-1] \* 9) % m # count as single digit from 1-9 or 11 to 19

elif s[i-1] == '2':

dp[i+1] = (dp[i+1] + dp[i-1] \* 6) % m

elif s[i-1] == "\*": # '\*\*'

dp[i+1] = (dp[i+1] + dp[i-1] \* 15) % m

else:

dp[i+1] = dp[i] if s[i] != '0' else 0

if s[i-1] == '1':

dp[i+1] = (dp[i+1] + dp[i-1]) % m

elif s[i-1] == '2' and int(s[i]) <= 6:

dp[i+1] = (dp[i+1] + dp[i-1]) % m

elif s[i-1] == '\*':

if int(s[i]) <= 6:

dp[i+1] = (dp[i+1] + 2 \* dp[i-1]) % m

else:

dp[i+1] = (dp[i+1] + dp[i-1]) % m

return dp[-1]

'''

check the current digit is '\*' or not,

if it is,count the comb when \* is one digit, which is 9

check the i-1 string, check if it is '1','2' or '\*';

do the same thing for other case

'''

637[Average of Levels in Binary Tree](https://leetcode.com/problems/average-of-levels-in-binary-tree/)55.7%Easy

class Solution(object):

def averageOfLevels(self, root):

"""

:type root: TreeNode

:rtype: List[float]

"""

if not root:

return []

queue = []

avg = []

queue.append(root)

while queue:

level\_num = len(queue)

node\_num = len(queue)

node\_sum = 0

while level\_num:

node = queue.pop(0)

node\_sum += node.val

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

level\_num -= 1

avg.append(float(node\_sum) / node\_num)

return avg

# [10,5,15,null,null,6,20]

# using bfs, get the value on each level and divide by node number

636[Exclusive Time of Functions](https://leetcode.com/problems/exclusive-time-of-functions/)40.3%Medium

class Solution(object):

def exclusiveTime(self, n, logs):

"""

:type n: int

:type logs: List[str]

:rtype: List[int]

"""

res = [0] \* n

stack = []

first = logs[0].split(":")

stack.append(int(first[0])) # push the start id to stack

i = 1

prev = int(first[2]) # record previous timestamp

while i < len(logs):

log = logs[i].split(':')

if log[1] == 'start':

if stack: # add time interval to last func in stack

res[stack[-1]] += int(log[2]) - prev

stack.append(int(log[0]))

prev = int(log[2])

else: # end

id = int(log[0])

res[id] += int(log[2]) - prev + 1 # +1 because end at very end

prev = int(log[2]) + 1

stack.pop()

i += 1

return res

'''

Use a stack to record all start timestamps of each function, and use

a variable prev to record last log time, use 'res' to record the time executed

for every func.

'''

621[Task Scheduler](https://leetcode.com/problems/task-scheduler/)41.8%Medium

'''

if the requirement is keeping the task order

'''

class Solution(object):

def leastInterval(self, tasks, n):

"""

:type tasks: List[str]

:type n: int

:rtype: int

"""

map = {}

cur = 0 # current position in the task line

for i in tasks:

if i not in map:

map[i] = cur + n + 1 # save cool time

cur += 1

else:

if cur < map[i]:

cur += (map[i] - cur + 1) # map[i] - cur for slots and 1 for itself

else:

cur += 1

map[i] = cur + n# update

return cur

class Solution(object):

def leastInterval(self, tasks, n):

"""

:type tasks: List[str]

:type n: int

:rtype: int

"""

count = [0] \* 26

for i in tasks:

count[ord(i) - ord('A')] += 1

count.sort()

maxVal = count[-1]

idleSlot = n \* (maxVal-1)

i = 24

while i >= 0 and count[i] > 0:

idleSlot -= min(count[i], maxVal-1) # should be smaller than maxVal-1

i -= 1

return idleSlot + len(tasks) if idleSlot > 0 else len(tasks)

'''

so the basic idea is to execute the most frequent task first, then count fixed

idle slots between it. Then check if the rest tasks can fit into slot. In the end

idle slot could be negative in this way(e.g n == 0), so it is either length of task

or the length plus positive idle slot number

time : O(n), loop over task using O(n), sorting takes O(26log(26)) = O(1)

'''

572[Subtree of Another Tree](https://leetcode.com/problems/subtree-of-another-tree/)40.6%Easy

class Solution(object):

def isSubtree(self, s, t):

"""

:type s: TreeNode

:type t: TreeNode

:rtype: bool

"""

return self.traverse(s,t)

def traverse(self, s, t):

return s != None and (self.equal(s,t) or self.traverse(s.left, t) or self.traverse(s.right, t))

def equal(self, s, t):

if not s and not t:

return True

if not s or not t:

return False

return s.val == t.val and self.equal(s.left, t.left) and self.equal(s.right, t.right)

554[Brick Wall](https://leetcode.com/problems/brick-wall/)44.9%Medium

class Solution(object):

def leastBricks(self, wall):

"""

:type wall: List[List[int]]

:rtype: int

"""

spaceMap = dict()

for row in wall:

spacelen = 0

for brick in row[:-1]: # avoid counting wall edge

spacelen += brick

if spacelen not in spaceMap:

spaceMap[spacelen] = 1

else:

spaceMap[spacelen] += 1

least = len(wall) - max(spaceMap.values()) if spaceMap.values() else len(wall) # avoid empty spaceMap for max

return least

s = Solution()

print s.leastBricks([[1,2,2,1],[3,1,2],[1,3,2],[2,4],[3,1,2],[1,3,1,1]])

'''

use spacemap to record where spaces exist in each row,

key is length(col number), value is number of spaces in all rows

'''

543[Diameter of Binary Tree](https://leetcode.com/problems/diameter-of-binary-tree/)43.6%Easy

class Solution(object):

def diameterOfBinaryTree(self, root):

"""

:type root: TreeNode

:rtype: int

"""

self.max = 0

self.height(root) # no need to get the return val, only focuses on the process

return self.max

# keep track of self.max in the process

def height(self, root):

if not root:

return 0

leftH = self.height(root.left)

rightH = self.height(root.right)

height = max(leftH, rightH) + 1

self.max = max(leftH + rightH, self.max) # not height

return height

# longest path is the sum of left tree height and right tree height

535[Encode and Decode TinyURL](https://leetcode.com/problems/encode-and-decode-tinyurl/)74.3%Medium

534[Design TinyURL](https://leetcode.com/problems/design-tinyurl/)0.0%Medium

525[Contiguous Array](https://leetcode.com/problems/contiguous-array/)39.4%Medium

class Solution(object):

def findMaxLength(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

prefix = {0: -1}

maxLength = 0

cumulative = 0

for i, val in enumerate(nums):

cumulative += 1 if val == 1 else -1

if cumulative in prefix:

maxLength = max(maxLength, i-prefix[cumulative])

else:

prefix[cumulative] = i

return maxLength

'''

it basically has the same idea of lc325, use a dictionary

tor record all prefix sum

'''

523[Continuous Subarray Sum](https://leetcode.com/problems/continuous-subarray-sum/)22.7%Medium

class Solution(object):

def checkSubarraySum(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: bool

"""

map = {0:-1}

s = 0

for i, val in enumerate(nums):

s += val

if k != 0:

s = s % k

if s in map:

if i - map[s] > 1:

return True

else:

map[s] = i

return False

'''

use a map to record previous mod and index. O(n)

'''

477[Total Hamming Distance](https://leetcode.com/problems/total-hamming-distance/)46.8%Medium

class Solution(object):

def totalHammingDistance(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

ans = 0

mask = 1

for i in xrange(32):

zeros = ones = 0

for num in nums:

if num & mask: # not num & mask == 1

ones += 1

else:

zeros += 1

ans += ones \* zeros

mask <<= 1

return ans

'''

only number of 1s and 0s on each bit matters, so we count ones and zeros,

add distance (ones \* zeros) to the result

'''

461[Hamming Distance](https://leetcode.com/problems/hamming-distance/)70.0%Easy

class Solution(object):

def hammingDistance(self, x, y):

"""

:type x: int

:type y: int

:rtype: int

"""

dif = x ^ y

count = 0

# count 1s in dif

while dif:

count += dif & 1

dif >>= 1

return count

# return bin(x^y).count('1')

410[Split Array Largest Sum](https://leetcode.com/problems/split-array-largest-sum/)37.1%Hard

404[Sum of Left Leaves](https://leetcode.com/problems/sum-of-left-leaves/)46.9%Easy

class Solution(object):

def sumOfLeftLeaves(self, root):

"""

:type root: TreeNode

:rtype: int

"""

self.sum = 0

self.dfs(root)

return self.sum

def dfs(self, root):

if not root:

return

if root.left:

if not root.left.left and not root.left.right: # left leaves

self.sum += root.left.val

else:

self.dfs(root.left)

if root.right:

self.dfs(root.right)

'''

left leaves, not left nodes

'''

275[H-Index II](https://leetcode.com/problems/h-index-ii/)34.4%Medium

265[Paint House II](https://leetcode.com/problems/paint-house-ii/) 38.0%Hard

261[Graph Valid Tree](https://leetcode.com/problems/graph-valid-tree/) 37.7%Medium

class Solution(object):

def validTree(self, n, edges):

"""

:type n: int

:type edges: List[List[int]]

:rtype: bool

"""

# should have no cycles

# should have no independent pairs

def find(nums, i):

if nums[i] == i: # is its own root

return i

return find(nums, nums[i])

nums = range(n)

for e in edges:

x = find(nums, e[0])

y = find(nums, e[1])

if x == y: # there is a cycle

return False

nums[x] = y # set the parent

# in the end, if there is no cycle and there n-1 edges, all node will be connected

return len(edges) == n-1

# time: O(v\*e)

168[Excel Sheet Column Title](https://leetcode.com/problems/excel-sheet-column-title/)26.0%Easy

import math

class Solution(object):

def convertToTitle(self, n):

"""

:type n: int

:rtype: str

"""

'''

eg. 52 = 26^1 \* 1 + 26^0 \* 26

'''

if n == 0:

return ""

right = chr((n - 1) % 26 + ord('A'))

return self.convertToTitle((n - 1) / 26) + right

80[Remove Duplicates from Sorted Array II](https://leetcode.com/problems/remove-duplicates-from-sorted-array-ii/)36.0%Medium

class Solution(object):

def removeDuplicates(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

i = 0

for n in nums:

if i < 2 or n > nums[i-2]: # if number is bigger than past 2 nums before i

nums[i] = n

i += 1

return i

'''

almost same idea as 19, but compare one more number.abs

Just go through the numbers and include those in the result that haven't been included twice already.

'''

26[Remove Duplicates from Sorted Array](https://leetcode.com/problems/remove-duplicates-from-sorted-array/)35.5%Easy

class Solution(object):

def removeDuplicates(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

if n < 2:

return n

i = 1

for j in xrange(1, n):

if nums[j] != nums[j-1]: # no duplicate

nums[i] = nums[j]

i += 1

return i # nums after i are duplicates

'''

move none duplicates forward

'''

not in tag:

65. Valid Number

class Solution(object):

def isNumber(self, s):

"""

:type s: str

:rtype: bool

"""

'''

http://www.cnblogs.com/grandyang/p/4084408.html

there are many cases that need to be considered:

1. space ' ': can only exist at the beginning or the end

2. dot '.': can only appear once, but it can be at front (".3") or middle(1.e2)

or end('1.'), but it can not be after 'e/E, like 2e.1(false), 1e1.1(false)

When it is at the end, the char before it should be digit, like "1."(true),

"-."(false)

3. 'e/E': it must have digits before or after it, and '.' can not be after it

4. '-/+': it can only be at the begining or right after 'e', like "+1.e+5" true

'''

num = False

numAfterE = True

dot = False

exp = False

sign = False

s = s.strip() # remove spaces for both end

for i in xrange(len(s)):

if s[i] == ' ':

return False # no space in the midlle

elif s[i] == '+' or s[i] == '-':

if i > 0 and s[i-1] != 'e': # if not after 'e'

return False

sign = True

elif s[i].isdigit():

num = True

numAfterE = True

elif s[i] == '.':

if dot or exp: # '.' or e appeard before

return False

dot = True

elif s[i] == 'e':

if exp or not num: # e appeared before or no number before it

return False

exp = True

numAfterE = False

else:

return False # orthe char

return num and numAfterE

s = Solution()

# print s.isNumber('.1')

print s.isNumber('1 ')

'''

# python trick

try:

float(s)

return True

except:

return False

'''

29. Divide Two Integers

class Solution(object):

def divide(self, dividend, divisor):

"""

:type dividend: int

:type divisor: int

:rtype: int

"""

maxInt = 2 \*\* 31 - 1

# corner case

if divisor == 0:

return maxInt

if dividend == 0:

return 0

sign = 1

if dividend > 0 and divisor < 0 or dividend < 0 and divisor > 0:

sign = -1

dividend = abs(dividend)

divisor = abs(divisor)

res = 0

c, sub = 1, divisor

while dividend >= divisor:

if dividend >= sub:

dividend -= sub

res += c

sub = (sub << 1)

c = c << 1

else:

sub = (sub >> 1)

c = c >> 1

res = res if sign == 1 else -res

return min(maxInt, max(-maxInt-1, res)) # avoid overflow here

s = Solution()

print s.divide(-3,3)

'''

for example, if we want to calc (17/2)

ret = 0;

17-2 ,ret+=1; left=15

15-4 ,ret+=2; left=11

11-8 ,ret+=4; left=3

3-2 ,ret+=1; left=1

ret=8;

the trick to accelerate the computing is to use a sub to get the divisor, and if it is smaller than divident,

multiply it by 2 by shifting to left everytime, otherwise shift back

Long division in binary:

The outer loop reduces n by at least half each iteration. So It has O(log N) iterations.

The inner loop has at most log N iterations.

So the overall complexity is O(( log N)^2)

'''

536. Construct Binary Tree from String

# Definition for a binary tree node.

# class TreeNode(object):

# def \_\_init\_\_(self, x):

# self.val = x

# self.left = None

# self.right = None

class Solution(object):

def str2tree(self, s):

"""

:type s: str

:rtype: TreeNode

"""

if not s:

return None

stack = []

i = 0

while i < len(s):

j = i # mark the starting point

c = s[i]

if c == ')': # node is used

stack.pop()

elif c.isdigit() or c == '-' :

while i + 1 < len(s) and s[i+1].isdigit(): # gather the number, e.g. '-12' or '123'

i += 1

curNode = TreeNode(int(s[j:i+1]))

if len(stack) != 0:

parent = stack[-1]

# check if left node is already occupied, otherwise add to right

if parent.left:

parent.right = curNode

else:

parent.left = curNode

stack.append(curNode)

i += 1

return stack[-1] # in the end, there is only root node left

bianry tree to circular list

'''

1

/ \

2 3

/ \ / \

4 5 6 7

-> <-4<->5<->2<->1<->3<->6<->7->

'''

class Node(object):

def \_\_init\_\_(self, value):

self.value = value

self.left = None

self.right = None

class Solution(object):

def concatenate(self, a, b):

if not a:

return b

if not b:

return a

# since it is a circular list

aEnd = a.left

bEnd = b.left

# concatenate head and tail

a.left = bEnd

aEnd.right = b

b.left = aEnd

bEnd.right = a

return a

def treeToList(self, n):

if not n:

return n

leftList = self.treeToList(n.left)

rightList= self.treeToList(n.right)

# make single node is self linked

n.left = n

n.right = n

# concatenate left subtree and right subtree

n = self.concatenate(leftList, n)

n = self.concatenate(n, rightList)

return n

'''

it is basically a modified in-order traverse of binary tree, every time

concatenate left subtree and parent node, parent node and right subtree.

In order to make circur link work, link the node to itself first.

'''

balance parenthesis in a string

'''

"(a)()" -> "(a)()"

"((bc)" -> "(bc)"

")))a((" -> "a"

"(a(b)" ->"(ab)" or "a(b)"

'''

def balanceParenthesis(s):

l, r = 0, 0

lscan = []

rscan = []

for i in s:

if i == '(':

l += 1

lscan.append('(')

elif i == ')':

if l > 0: # has match '(' before it

l -= 1

lscan.append(')')

else:

lscan.append(i)

for i in lscan[::-1]:

if i == ')':

r += 1

rscan.append(')')

elif i == '(':

if r > 0:

r -= 1

rscan.append('(')

else:

rscan.append(i)

return ''.join(rscan[::-1])

print balanceParenthesis("(a(b)")

'''

2 rounds of scans with a counter on open parens first and one with a counter on close parens. The first scan finds all unmatched close parens and the second one finds all unmatched open parens

'''